Scripts Use for the Synthesis and the Simulation of VHDL Circuits and Evaluation of the Power Consumption

Romain Michard

INRIA

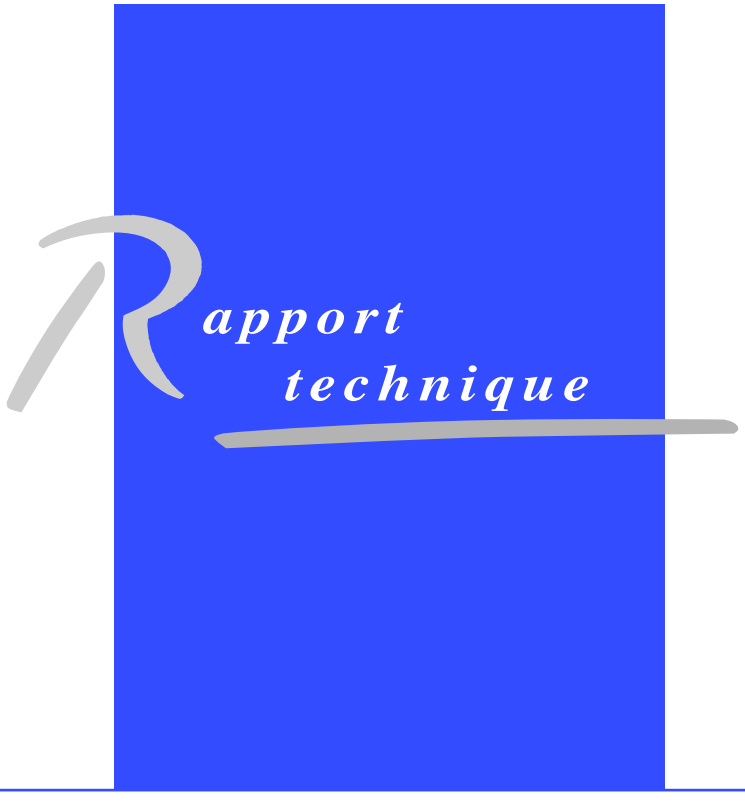
INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Scripts Use for the Synthesis and the Simulation of VHDL Circuits and Evaluation of the Power Consumption

Romain Michard

N° 0381 — version 2

initial version Février 2010 — revised version Mars 2010



ISSN 0249-0803 ISRN INRIA/RT--0381--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
GRENOBLE - RHÔNE-ALPES

Scripts Use for the Synthesis and the Simulation of VHDL Circuits and Evaluation of the Power Consumption

Romain Michard*

Thème : COM
Équipe-Projet Swing

Rapport technique n° 0381 — version 2 — initial version Février 2010 — revised version Mars 2010 — 20 pages

Abstract: This document presents a way of using scripts for the synthesis or the simulation of VHDL hardware components. Moreover it explains the estimation of the power consumption of such circuits that can be evaluated by XPower. The use of such scripts allows to automate the procedures and to deal with many circuits without repeating the same tasks several times.
The steps presented in this document are based on scripts that are published under the GNU GPL, they can be used and modified as wanted by the user.

Key-words: VHDL, circuit, synthesis, simulation, script, automation, consumption, XPower

* INRIA, INSA Lyon, CITI, F-69621, France - romain.michard@inria.fr

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Utilisation de scripts pour la synthèse et la simulation de circuits VHDL et évaluation de la puissance électrique consommée

Résumé : Ce document décrit une manière dont on peut utiliser des scripts pour synthétiser ou simuler des composants matériels écrits dans le langage VHDL. Il explique aussi l'estimation de la consommation de puissance électrique de ce genre de circuits qui peut être réalisée par l'outil XPower. L'utilisation de tels scripts permet d'automatiser les différentes procédures et de réaliser le traitement d'une grande quantité de circuits sans avoir à répéter les mêmes tâches plusieurs fois.
Les étapes présentées dans ce document sont toutes basées sur des scripts publiés sous GNU GPL, ils peuvent être utilisés et modifiés à volonté selon les souhaits de l'utilisateur.

Mots-clés : VHDL, circuit, synthèse, simulation, script, automatisation, consommation, XPower

Contents

License

This software is to be used or modified as wanted, the writer won't assume any maintenance nor any help, however a message to tell it's used would be appreciated.

1 Introduction

This report is written to make scripts for synthesis, simulation and power consumption estimation of VHDL circuits easy to understand and to use. Graphical user interfaces (GUIs) are generally used to implement hardware designs from hardware description language (HDL) files. For example all the software tools used in this report are available in GUI modes: Xilinx® ISE Project Navigator or XPower Analyzer, Mentor Graphics® ModelSim, etc.

The scripts are in different languages depending on what seems to be the best to achieve the aim. For example the synthesis script is in the Perl language because it's a language that makes scripting easy, the simulation script is written in Tcl because it's a language understood by ModelSim.

For convenient matters the system used is Windows® (because Xilinx® ISE is available on a Windows® platform in this case) and the scripts are launched in a Cygwin® terminal so as to make the Linux® commands (as `grep` for example) available.

This paper is organised as follows: Section 2 describes how the whole synthesis process is handled, Section 3 is explaining the simulation, the power consumption estimation is presented in Section 4, finally a conclusion is given in Section 5.

2 Synthesis

The script presented on Figure 1 is used to synthesize a VHDL source file. The source file is a very simple adder presented on Figure 4. The size of the operands (the inputs and the output) is 12 bits but it could be configure as anything else.

All the help for this script can be read on the Xilinx® website [6] or in the application folder (`$installation_directory$\ISE\doc\usenglish\isehelp\`) in several `.pdf` files as `cgd.pdf, devref.pdf, manuals.pdf, sim.pdf, xst.pdf,...`

Here is a description of the synthesis script:

Lines 23 to 44 The main procedure is running the different subroutines depending on the options.

Lines 48 to 77 The `synthesis` subroutine is generating another script with all the options that will be run by `xst` to synthesize the component.

Lines 69 to 75 That prints a report of the synthesis.

Lines 81 to 101 The `translate` subroutine produces a Native Generic Object (`.ngo`) after the synthesis.

Lines 105 to 127 The `mapping` subroutine maps the logical functions on the available hardware resources of the FPGA.

Lines 131 to 143 The `placeroute` subroutine is easily understood. It places the hardware resources and routes wires where they are necessary.

Lines 147 to 156 The `netgen` subroutine back-annotates the logical design depending on the first synthesis process in order to obtain a physical post-place&route model that can be accurately simulated.

Lines 160 to 179 The `extract` subroutine uses the `trce` program to generate a static timing report of the post-place&route model. All those extracted values can be used by the `report` subroutine.

Lines 183 to 214 The `report` subroutine creates a LaTeX file reporting whatever the user is interested in.

Lines 218 to 272 The `parsecmdline` subroutine gets the different command-line options and checks whether everything is correct for the programs to run.

Lines 276 to 281 The `getinputenv` subroutine gets the VHDL file name and creates all the different environment variables depending on it.

One can use a shell script as the one presented on Figure 5 to synthesize several source files (all the `.vhd` files of the `src` directory in this case) with only one command. This script uses a device list as the one presented on Figure 2 where everything is commented by a # except the line with the targeted FPGA.

The synthesis script generates a report file for each circuit. It would be very fastidious to read these separate files. Fortunately the Perl script of Figure 3 is able to merge several LaTeX files into one.

Separate tables are merged into fewer ones of 50 lines. This program won't be detailed because of its simplicity; it's only text reporting matters and not a technical point.

3 Simulation

All the help for this script can be read on the ModelSim website [3] or in the application folder (`$installation_directory$\docs\pdfdocs\`) in several `.pdf` files.

To simulate a component (whatever the model is) we use this command:

```
vsim -c -do $simulation_file$
```

This command is still to be launched in a Cygwin® terminal so as to make the classical commands available.

The `$simulation_file$` variable can be changed to simulate different environments. For this report two different simulations are executed: first the behavioral model, to ensure it's correct, then the post-place&route one to get a signals activity report in order to evaluate the power consumption accurately. To make it easier to handle, the script is cut into two parts: the first one is dealing with the component to simulate, the second one is generating the test vectors as the same test is executed for both simulations.

3.1 Behavioral model

When designing a VHDL model of a circuit it's important to simulate it all along the design process in order to be sure of its proper behavior. To simulate it the script represented on Figure 6 is used with the test vectors of `vect.tcl` presented on Figure 8. Here is a description of this script:

Lines 15 to 17 The script is defining different directory variables. As the script is run in a Windows® system, paths are written C:... but the \ is replaced by a /.
Line 19 The script is printing some information for the user to be aware of the running process.
Line 22 The working library is defined.
Line 25 The components are compiled.
Line 28 The `my_adder` component is simulated.
Lines 31 to 34 Some signals are added to the visualisation list.
Line 37 The simulation vectors of `vect.tcl` are executed.
Line 40 Another information printing.
Line 41 The simulation is terminating.

3.2 Post-place&route model

It could be interesting to simulate the post-place&route model. The main utility is to generate the signals activity report file (`.saif`) for the power consumption estimation to be accurate. The structure of the script is the same as the one for the behavioral model. The differences are described in the following:

Line 25 The compiled file is `circ.sim.vhd` (in this case) generated by the place&route process.
Line 37 The `.sdf` file is giving information on the delays in the circuit, it's generated by the place&route process and is necessary for this simulation.
Line 38 Every ports and internal signals are looked at to know the associated activity.
Line 43 The output activity report file is generated in an SAIF format now the simulation is done.

At this point a signal setup time violation error could happen, the reason is not completely clear and the best way to avoid it was to change the targeted FPGA back in the synthesis step (for example a Virtex4 instead of a Spartan3).

3.3 Test vectors

The test vectors are written in a separate file as it's simpler to use the same vectors for both the behavioral and the post-place&route models. The script of Figure 8 can be described as follows:

Line 17 A parameter is defined for the size of the signals.

Lines 20 to 43 Several conversion functions are defined. All are not used for this simulation (as the `bits2int` function or the `bin2dec` one) but they could be helpful for others.

Lines 46 to 53 A mathematical function is defined to easily compute the power of a number.

Lines 56 to 66 The simulation loop is controlling the input signals during the simulation.

Line 65 A report file is generated to check whether everything is going as expected.

4 Power consumption estimation

Now the `.saif` file is generated, an accurate power consumption evaluation is available by Xilinx® XPower if a constraint file (`.pcf` file provided by the `map` program) is provided. To run this estimation this command can be run:

```
xpwr circ.par.ncd -s circ_xpower.saif circ.pcf
```

The `circ.par.ncd` and `circ_xpower.saif` files are generated by the previous steps.

This estimation produces a report that is printed on the standard output and in the `circ.par.pwr` file.

Reducing the power consumption is something that really matters nowadays and it should be an important effort when designing hardware circuits. This method allows to know how much power the circuit could consume. It's only an estimation of this consumption but tools seem very accurate today, they have improved a lot in the past years and the accuracy is increasing. The best would obviously be to build the hardware circuit and to measure its consumption in real conditions but it would be more painful to do and it would ask some tools (as a hardware platform for the benchmarks and measures,...). The solution presented in this document is a good trade-off to evaluate the consumption of a circuit with few needs.

5 Conclusion

This paper explains a scripted solution to simulate, synthesize and estimate the power consumption of a VHDL source circuit description. This method is not the only one. For example a solution named Athena [5] exists, it is created for cryptographic computing circuits when the presented method is more generic and it needs the installation of a complete program when this one only uses the standard CAD tools. Furthermore it is only synthesizing the circuit, no simulation nore any power consumption are available.

All the websites helping for that matter are listed in the References section.

This report is to be modified or completed if other precisions seem to be important to note or if changes happen.

References

[1] Tcl Developer Xchange. `http://www.tcl.tk`.

[2] CPAN. The Perl Programming Language. `http://www.perl.org`.

[3] Mentor Graphics®. Modelsim® website. `http://model.com`.

[4] Red Hat. Cygwin website. `http://www.cygwin.com`.

[5] CERG: Cryptographic Engineering Research Group of the George Mason University. Athena. `http://cryptography.gmu.edu/athena/`.

[6] Xilinx®. ISE design tools documentation website. `http://www.xilinx.com/tools/designtools.htm`.

Appendix

A Synthesis

This script is represented on Figure 1, it's based on what Arnaud Tisserand wrote and is written in the Perl language. I would like to really thank Arnaud for what he did.

B Simulation

These scripts are Tcl ones because it's the language used by Modelsim.

```
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see http://www.gnu.org/licenses/.

#!/usr/bin/env perl

my $device, $optim_target, $optim_effort;
my $filename, $prefix, $path, $suffix;
my $mult_style;
my $per, $size;

# ------------------------------------------------------------------------

parsecmdline();
getinputenv();

if ($opt_v) {
    print "device: ", $device, ", optimization: ", $optim_target,
          ", effort: ", $optim_effort, ", mult_style: ", $mult_style,
          "\n";
    if ($opt_p) {print "clock period constraint: ", $opt_p, "\n";}
}

synthesis();    # x.vhdl => x.ngc (report xst.srp)
translate();    # x.ngc => x.ngd (report x.bld)
mapping();              # x.ngd => x.map.ncd (report x.map.mrp)
placeroute();   # x.map.ncd  => x.par.ncd (report x.par)
netgen();               # x.par.ncd => x.sim.vhd
extract();              # x.par.ncd => x.twr (<-report)

if ($opt_t) {
       report();
}
system "rm -rf *.bld *.ucf *.twr *.mrp *.ngm *.pad *.par *.ngc *.ngd *.xpi *.unroutes *.csv *.xml *.lst *.srp *.xrpt
*.script xst *.map *.ptwx *.nlf *.twx *pad.txt xlnx_* *.map.ncd";
exit(0);

# ------------------------------------------------------------------------

sub synthesis {
    my $script = "run\n";
    $script .= "-ifn $ARGV[1]\n-ifmt VHDL\n";
    $script .= "-ofn $prefix.ngc\n-ofmt NGC\n";
    $script .= "-p $device\n";
    $script .= "-opt_mode $optim_target\n";
    $script .= "-mult_style $mult_style\n";
    if ($optim_effort =~ /high/i) {$script .=  "-opt_level 2\n";}
    else {$script .= "-opt_level 1\n";}

    open xstscript, "> xst.script" or die;
    print xstscript "$script\n";
    close xstscript;

    my $cmd = "xst.exe -ifn xst.script";
    system $cmd;

    if ($opt_v) {
                print "----- synthesis -----\n";
                print "xst: $script\n";
                print "cmd: $cmd\n";
                open report, "< xst.srp" or die;
                while (<report>) {
                        if (/number of slices/i) { print $_; }
                        if (/number of mult18x18s/i) { print $_; }
                        if (/minimum period/i) { print $_; }
```

```
                }
                close report;
    }
}

# ------------------------------------------------------------------------

sub translate {
    my $cmd = "ngdbuild.exe -p $device";
    if ($opt_p) {
                open ucffile, "> ./$prefix.ucf" or die;
                print ucffile "NET \"clk\" TNM_NET = \"clk\";\n";
                print ucffile "TIMESPEC \"TS_clk\" = PERIOD \"clk\" $opt_p ns HIGH 50 %;\n";
                close ucffile;
                $cmd .= " -intstyle ise -uc $prefix.ucf";
    }
    else {
                $cmd .= " -i";
    }

    $cmd .= " $prefix.ngc";

    if ($opt_v) {
                print "----- translate -----\n";
                print "cmd: $cmd\n";
        }
    system $cmd;
}

# ------------------------------------------------------------------------

sub mapping  {
    my $cmd = "map.exe  -p $device";
    $cmd .= " -o $prefix.map.ncd $prefix.ngd";
    if ($opt_p) {$cmd .= " $prefix.pcf";}

    if ($opt_v) {
                print "----- mapping -----\n";
                print "cmd: $cmd\n";
        }
    system $cmd;

    open report, "< ./$prefix.map.mrp" or die;
    while (<report>) {
                if (/number of occupied slices/i) {
                        print $_;
                        s/^[ \t]*number of occupied slices:[ \t]*//i;
                        s/ out.*\n//i;
                        $size = $_;
                }
                if (/number of mult18x18s/i) {print $_;}
    }
    close report;
}

# ------------------------------------------------------------------------

sub placeroute {
    my $cmd = "par.exe  -w";
    if ($optim_effort =~ /high/i) {$cmd .= " -ol high";}
    else {$cmd .= " -ol std";}
    $cmd .= " $prefix.map.ncd $prefix.par.ncd";
    if ($opt_p) { $cmd .= " $prefix.pcf"; }

    if ($opt_v) {
                print "----- place&route -----\n";
                print "cmd: $cmd\n";
        }
    system $cmd;
}

# ------------------------------------------------------------------------

sub netgen {
```

```
    my $cmd = "netgen.exe -w -rpw 100 -tpw 0 -sim -ofmt vhdl";
    $cmd .= " -pcf $prefix.pcf $prefix.par.ncd $prefix.sim.vhd";

    if ($opt_v) {
                print "----- netgen -----\n";
                print "cmd: $cmd\n";
        }
    system $cmd;
}

# ------------------------------------------------------------------

sub extract {
    my $cmd = "trce.exe  $prefix.par.ncd";
    if ($opt_p) {$cmd .= " $prefix.pcf";}
    if ($opt_v) {
                print "----- extract -----\n";
                print "cmd: $cmd\n";
        }
    system $cmd;

    open report, "< ./$prefix.par.twr" or die "error opening report file : $!\n";
    while (<report>) {
                if (/minimum period/i) {
                        print $_;
                        s/^[ \t]*minimum period:[ \t]*//i;
                        s/ns.*//i;
                        $per = ceil($_*10)/10;
                }
    }
    close report;
}

# ------------------------------------------------------------------

sub report {
        my $rep_file="$prefix.tex";
        print "Reporting in $rep_file\n";
        open rep, "> $rep_file";
    my $preamble = <<EOF;
\\documentclass{article}
\\usepackage{fullpage}

\\begin{document}

\\begin{table}[!ht]
\\begin{center}
\\begin{tabular}{|*{3}{r|}}
\\hline
\\textbf{Circuit}&\\textbf{period}&\\textbf{slices}\\\\
\\hline
\%-----start data
EOF
    my $str ="$prefix&$per&$size\\\\\n\\hline\n";
    my $end = <<EOF;
\%-----end data
\\end{tabular}
\\end{center}
\\end{table}
\\end{document}
EOF

        print rep $preamble;
    print rep $str;
        print rep $end;
        close rep;
}

# ------------------------------------------------------------------

sub parsecmdline {
        use POSIX;
    use Getopt::Std;
```

```
    getopts("p:m:tsovx"); # parse the command line options

    my $usage = <<EOF;
Usage: xlnx [-p period] [-s] [-o] [-m style] [-v] [-x] [-t] device_file vhdl_fil
e
Description:
        apply Xilinx synthesis (XST) and implementation tools
        on the vhdl_file with the device in device_file
Options:
        -p              constraint period [ns]
        -s              speed optimization (default is area)
        -o              high optimization effort (default is low)
        -m              multiplieur style [auto, block, lut, pipe_lut] (default
is auto)
        -v              verbose mode (all commands and intermediate results)
        -t              report results in a texfile
EOF

    # 2 mandatory args: device_file (arg 0) vhdl_file (arg 1)
    if ($#ARGV != 1) {die $usage;}

    # check devicefile
    -e $ARGV[0] or die "device file $ARGV[0] not found\n";

    # check vhdlfile
    -e $ARGV[1] or die "vhdl file $ARGV[1] not found\n";

        # set device
    open device, "< $ARGV[0]" or die;
    while (<device>) {
                if ((!/^[ \t]*\#/i) && (!/^[ \t]*$/i)) {
                        s/^[ \t]*//;
                        s/[ \t\n]*$//;
                        $device = "$_";
                }
    }
    close device;

    # period option
    if ($opt_p) {
                if ($opt_p < 1) {die "not possible clock period\n";}
    }

    # optimization options
    if ($opt_s) {$optim_target = "speed";}
    else {$optim_target = "area";}

    if ($opt_o) {$optim_effort = "high";}
    else {$optim_effort = "low";}

    if ($opt_m) {$mult_style = $opt_m;}
    else {$mult_style = "auto";}
}

# ------------------------------------------------------------------------

sub getinputenv {
    use File::Basename;

        $filename = $ARGV[1];
    ($prefix,$path,$suffix) = fileparse($filename,"\.vhd?");
}
```

Figure 1: Perl script for the synthesis

```
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see http://www.gnu.org/licenses/.

#ISE 11.3

# Spartan 3A (sg: -4, -5)
#xc3s50a-ft256-4
#xc3s200a-ft256-4
#xc3s400a-ft256-4
#xc3s700a-ft256-4
#xc3s1400a-ft256-4

# Spartan 3AN (sg: -4, -5)
#xc3s50an-tqg144-4
#xc3s200an-ftg256-4
#xc3s400an-fgg400-4
#xc3s700an-fgg484-4
#xc3s1400an-fgg676-4

# Spartan 3E (sg: -5, -4)
#xc3s100e-cp132-4
#xc3s250e-ft256-4
#xc3s500e-ft256-4
#xc3s1200e-ft256-4
#xc3s1600e-fg320-4

# Spartan 3 (sg: -5, -4)
xc3s50-pq208-4
#xc3s200-ft256-4
#xc3s400-ft256-4
#xc3s1000-ft256-4
#xc3s1500-fg456-4
#xc3s2000-fg456-4
#xc3s4000-fg676-4
#xc3s5000-fg676-4
#xc3s1000l-fg456-4
#xc3s1500l-fg456-4
#xc3s4000l-fg900-4

# Spartan 6 (sg: -3, -2)
#xc6slx9-ftg256-2
#xc6slx16-ftg256-2
#xc6slx25-ftg256-2
#xc6slx45-fgg676-2
#xc6slx75-csg484-2
#xc6slx100-csg484-2
#xc6slx150-csg484-2
#xc6slx25t-fgg484-2
#xc6slx45t-fgg484-2
#xc6slx75t-fgg676-2
#xc6slx100t-fgg676-2
#xc6slx150t-fgg676-2

# Virtex 4 (sg: -12, -11, -10)
#xc4vlx15-ff668-11
#xc4vlx25-ff668-11
#xc4vlx40-ff668-11
#xc4vlx60-ff668-11
#xc4vlx80-ff1148-11
#xc4vlx100-ff1148-11
#xc4vlx160-ff1148-11
#xc4vlx200-ff1513-11
#xc4vsx25-ff668-11
#xc4vsx35-ff668-11
```

```
#xc4vsx55-ff1148-11
#xc4vfx12-ff668-11
#xc4vfx20-ff672-11
#xc4vfx40-ff672-11
#xc4vfx60-ff672-11
#xc4vfx100-ff1517-11
#xc4vfx140-ff1517-11

# Virtex 5 (sg: -3, -2, -1)

#xc5vlx30-ff676-2
#xc5vlx50-ff676-2
#xc5vlx85-ff676-2
#xc5vlx110-ff676-2
#xc5vlx155-ff1760-2
#xc5vlx220-ff1760-2
#xc5vlx330-ff1760-2
#xc5vlx20t-ff323-2
#xc5vlx30t-ff323-2
#xc5vlx50t-ff1136-2
#xc5vlx85t-ff1136-2
#xc5vlx110t-ff1136-2
#xc5vlx155t-ff1136-2
#xc5vlx220t-ff1738-2
#xc5vlx330t-ff1738-2
#xc5vsx35t-ff665-2
#xc5vsx50t-ff665-2
#xc5vsx95t-ff1136-2
#xc5vsx240t-ff1738-2
#xc5vfx30-ff665-2
#xc5vfx70-ff665-2
#xc5vfx100-ff1738-2
#xc5vfx130-ff1738-2
#xc5vfx200-ff1738-2
#xc5vtx150-ff1759-2
#xc5vtx240-ff1759-2

# Virtex 6 (sg: -3, -2, -1)

#xc6vlx75t-ff784-2
#xc6vlx130t-ff784-2
#xc6vlx195t-ff784-2
#xc6vlx240t-ff784-2
#xc6vlx365t-ff1759-2
#xc6vlx550t-ff1759-2
#xc6vlx760-ff1760-2
#xc6vsx315t-ff1156-2
#xc6vsx475t-ff1156-2
#xc6vhx250t-ff1154-2
#xc6vhx255t-ff1155-2
#xc6vhx380t-ff1155-2
#xc6vhx565t-ff1923-2
#xc6vcx75t-ff784-2
#xc6vcx130t-ff784-2
#xc6vcx195t-ff784-2
#xc6vcx240t-ff784-2
```

Figure 2: Text with the different available FPGA architectures

```
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see http://www.gnu.org/licenses/.

#!/usr/bin/env perl

# ----------------------------------------------------------------------
system "ls *.tex > tex.lst";
open lst, "<tex.lst";
open out, ">syn-merge.tex";
my $nb = 0;
my $cond = 0;
my $rep;
begin();
while(<lst>){
        if($_ =~ m/[a-z].*/ and $_ !~ m/syn-merge.tex/){
                $rep = $_;
                print "$nb $rep";
                open rep, "< $rep";
                while(<rep>){
                        if($_ =~ m/%-----s/){
                                $cond = 1;
                        }
                        elsif($cond == 1 and $_ =~ m/%-----e/){
                                $cond = 2;
                        }
                        elsif($cond == 1){
                                print out $_;
                        }
                }
                close rep;
                $nb++;
                if($nb == 51){
                        $nb = 1;
                        cut();
                }
        };
}
end();
close lst;
close out;
exit(0);

# ----------------------------------------------------------------------
sub begin
{
    my $preamble = <<EOF;
\\documentclass{article}
\\usepackage{fullpage}

\\begin{document}

\\begin{table}[!ht]
\\begin{center}
\\begin{tabular}{|*{3}{r|}}
\\hline
\\textbf{circuit}&\\textbf{period}&\\textbf{slices}\\\\
\\hline
EOF

        print out $preamble;
}

# ----------------------------------------------------------------------
sub end
```

```
{
    my $end = <<EOF;
\\end{tabular}
\\end{center}
\\end{table}

\\end{document}
EOF

        print out $end;
}

# ------------------------------------------------------------------
sub cut
{
    my $cut = <<EOF;
\\end{tabular}
\\end{center}
\\end{table}


\\begin{table}[!ht]
\\begin{center}
\\begin{tabular}{|*{4}{r|}}
\\hline
\\textbf{circuit}&\\textbf{period}&\\textbf{slices}\\\\
\\hline
EOF

        print out $cut;
}
```

Figure 3: Perl programm for the reports merging

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity my_adder is
        port(   x : in std_logic_vector(11 downto 0);
                y : in std_logic_vector(11 downto 0);
                clk : in std_logic;
                q : out std_logic_vector(11 downto 0));
        end entity;
architecture arch of my_adder is
        signal x1 : std_logic_vector (11 downto 0);
        signal x2 : std_logic_vector (11 downto 0);
        signal y1 : std_logic_vector (11 downto 0);
        signal y2 : std_logic_vector (11 downto 0);
        signal q1 : std_logic_vector (11 downto 0);
        signal q2 : std_logic_vector (11 downto 0);
begin
        inReg: PROCESS(clk,x)
        BEGIN
                IF clk'event AND clk='1' THEN
                        x1<=x;
                        x2<=x1;
                        y1<=y;
                        y2<=y1;
                        q1<=q2;
                        q<=q1;
                END IF;
        END PROCESS inReg;
q2<=x2+y2;
end architecture;
```

Figure 4: The VHDL source file

```
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see http://www.gnu.org/licenses/.

#!/usr/bin/sh
for i in src/*.vhd; do
echo $i;
perl xlnx.pl -p 10 -v -o -m lut -t device.txt $i;
done;
```

Figure 5: The shell script

```
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see http://www.gnu.org/licenses/.

#directories definition
set PROOT E:/Work/INSA/synchro/work/Divers/scripts/sim
set PVHDL $PROOT
set PSIM  $PROOT

puts "Simulation starting"

#work library definition
vlib $PSIM/work

#components compilation
vcom -work $PSIM/work $PVHDL/circ.vhd

#component simulation
vsim my_adder

#visualisation signals
    add list -unsigned clk
    add list -unsigned x
    add list -unsigned y
    add list -unsigned q

#simulation vectors
    do vect.tcl

#end
puts "Simulation ending"
quit -f
```

Figure 6: Tcl script for the simulation of the behavioral model

```
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see http://www.gnu.org/licenses/.

#directories definition
set PROOT E:/Work/INSA/synchro/work/rt-scripts/sim
set PVHDL $PROOT
set PSIM  $PROOT

puts "Simulation starting"

#work library definition
vlib $PSIM/work

#components compilation
vcom -work $PSIM/work $PVHDL/circ.sim.vhd

#component simulation
vsim my_adder

#visualisation signals
    add list -unsigned clk
    add list -unsigned x
    add list -unsigned y
    add list -unsigned q

#for XPower
vsim -t 1ps -sdfmax "/my_adder=circ.sim.sdf"  -lib work my_adder
power add -ports -internal *

#simulation vectors
    do vect.tcl

power report -bsaif $PROOT/circ_xpower.saif

#end
puts "Simulation ending"
quit -f
```

Figure 7: Tcl script for the simulation of the post–place&route model and the generation of the SAIF file

```
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see http://www.gnu.org/licenses/.

#test vectors

#parameters
set x_size 12

#conversion fonctions
proc int2bits {i {digits {} }} {
        binary scan [binary format I1 $i] B* x
        set len [string length $x]
        set low [expr $len-$digits]
        set bitvalue [string range $x $low $len]
}

proc bits2int {bits} {
        set res 0
        foreach i $bits {
                set res [expr {$res*2+$i}]
        }
        set res
}

proc bin2dec {num} {
        set num h[string map {1 i 0 o} $num]
    while {[regexp {[io]} $num]} {
       set num\
         [string map {0o 0 0i 1 1o 2 1i 3 2o 4 2i 5 3o 6 3i 7 4o 8 4i 9 ho h hi
h1}\
           [string map {0 o0 1 o1 2 o2 3 o3 4 o4 5 i0 6 i1 7 i2 8 i3 9 i4} $num]
]
    }
    string range $num 1 end
}

#power fonction
proc pwer {base p} {
    set result 1
    while {$p > 0} {
        set result [expr $result * $base]
        set p [expr $p - 1]
    }
    return $result
}

#simulation
for {set x 1} {$x<[pwer 2 $x_size]} {set x [expr $x+111]} {
        puts  $x
        force x [int2bits $x 12]
        force y [int2bits [expr {$x * 2}] 12]
        force clk 0
        run 5 ns
        force clk 1
        run 5 ns

        write list res.lst
}

#end
```

Figure 8: Tcl script to generate the test vectors

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803